Torque control for Staubli

Overview

This program controls the TX90 robot using a PID controller and two types of trajectory planning (Quintic and LSPB). The robot is controlled via torque commands and the compensation of gravity and dynamics is applied. To use this code some libraries are need as well as Matlab imported source code for the dynamic regression functions.

- Eigen/Dense: C++ template library for linear algebra
- TX90Dynamics.cpp
    - Contains all the matrixes of the regression functions. Imported from matlab

The Program uses the next header files:

- StaubliUdpPort.h: UDP communication with the robot
- TX90ProxyByTorque.h: Proxy class for communication with the robot
- TrajQuinticPVA.h: Trajectory planning library for Quintic trajectories
- PIDCtrlPVA.h: PID class that implements a PID controller for the robot
- TrajLSPBPVA.h: Trajectory planning library for LSPB trajectories
- DynamicCompesator.h: Class that is used to calculate compensates the robot's dynamics from matlab functions
- HystoricData.h: Records data to .txt

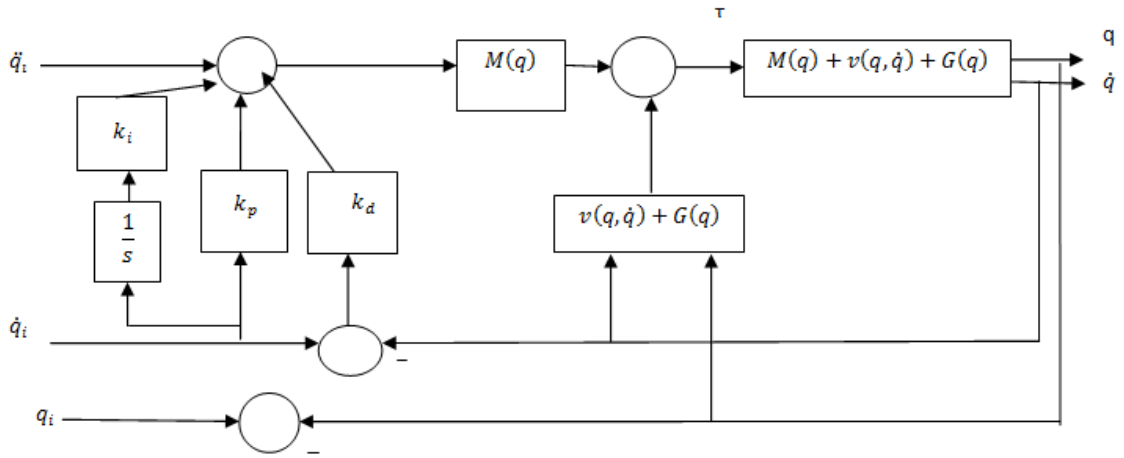
The code defines the function "fillVectorOfControllerWithQuinticTraj", which fills an array of PID controllers with a Quintic trajectory for each joint.

The main function starts by creating instances of the necessary classes:

- TX90ProxyByTorque: A proxy class for communication with the robot
- PIDCtrlPVA: A PID controller for each joint
- DynamicCompesator: A class that compensates the robot's dynamics
- HystoricData: A class that records data during execution

CONTROL EXPLANATION

This code is the implementation of a torque control in joint state for the TX90 robot. The following scheme represents the control.



In this control strategy, the desired joint angle, velocity, and acceleration (or torque) are specified as input to the controller.

The PID controller computes an error signal as the difference between the desired and actual joint angle, velocity, and acceleration. This error signal is then used to calculate the control output summing the proportional, derivative and integral gains.

The dynamic compensation is calculated using the Coriolis and centrifugal components v(q,q) and the gravitational term g(q).

Finally, the computed torque, which is the sum of the PID control output and the feedforward torque is applied to the robot joint.

CODE EXPLANATION (MainControl.cpp)

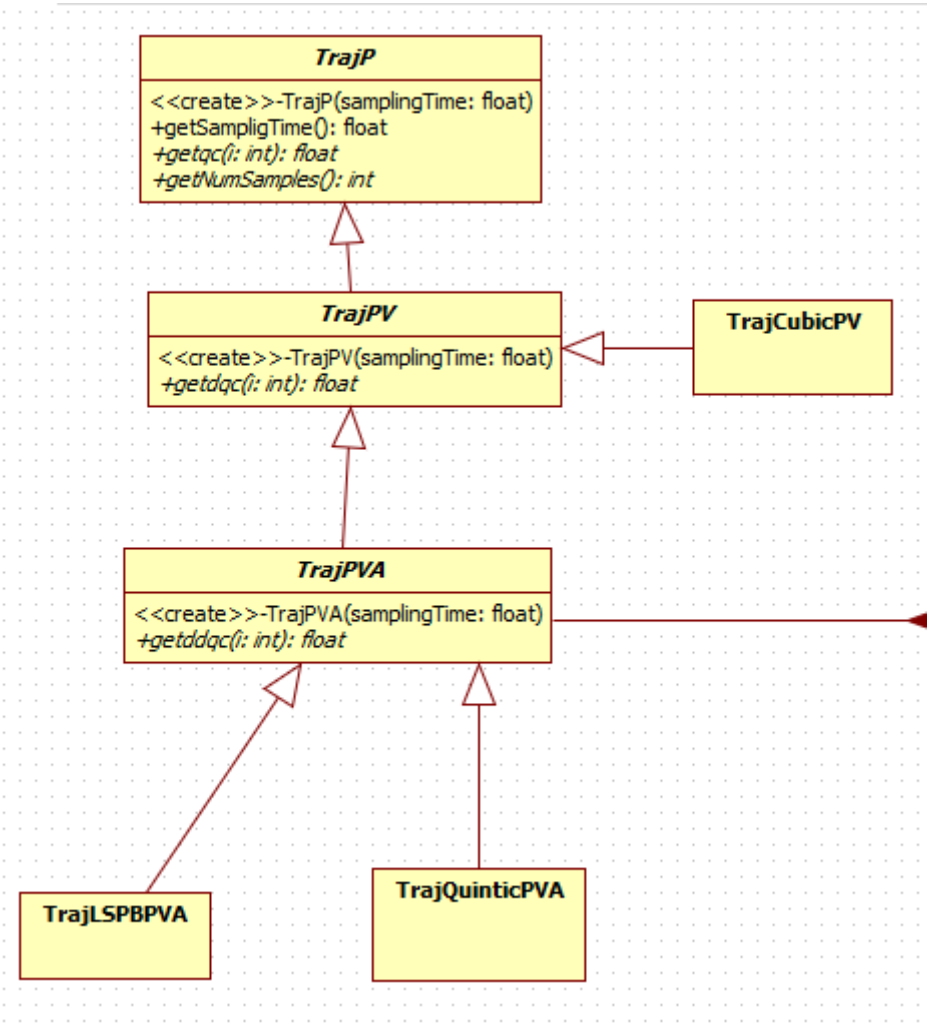
First, some of the variables and for the control are initialized such as the time step, gains for the PID, initial and final position and the constant speed (in case of using the LSPB trayectory)

```
//Time step
float const deltaT = 0.004;
float const TrajTime = 10;

//PID GAINS
std::vector<float> const Kp = { 1200,   600,    60,   600,    700,    900 };
std::vector<float> const Kv = { 20,     10,     12,     10,     13,     17 };
std::vector<float> const Ki = { 0.1,    0.1,    0.35,   0.4,    0.4,    0.7 };

// Initial and final position for robot axis
std::vector<float> const posI = { 0,    0,      0,      0,    0,    0 };
std::vector<float> const posF = { 1.57,   -0.78 ,     1.57,     1.57  ,   1.57,   1.57 };

// LSPB trayectory constant speed
std::vector<float> velConsts = { 0.19,0,0,0,0,0 };
```

For the trajectory generation, an abstract VirtualTrajTypes.h has been created in order to gather different trajectories that shares methods and attributes. There are 3 class inside:

- TrajP: main class with three methods
  - getsamplingTime: returns the controls time step
  - getqc: returns the desired trajectory position value for a determined iteration
  - getNumSamples: returns the size of the desired trajectory size
- TrajPV: this class is inherited from TrajP and adds a new method
  - Getdqc: returns the desired trajectory velocity value for a determined iteration.
- TrajPVA: this call is inherited from TrajPV and adds the last method
  - Getddqc: returns the desired trajectory acceleration value for a determined iteration

In this proyect there are 2 different trajectories as said before, quintic and LSPB. So, as both compute the position, velocity and acceleration for the trajectories, then both *TrajQuinticPVA* and *TrajLSPBPVA* classes inherited from the TrajPVA class. The image below shows how does it work.



Returning to the main code “MainControl.cpp”, the function "fillVectorOfControllerWithQuinticTraj" and “fillVectorOfControllerWithLSPBTraj” are called to fill the array of PID controllers with a Quintic or the LSBP trajectories.

```
//fills the array of PID controllers for both trayectories
void fillVectorOfControllerWithQuinticTraj(PIDCtrlPVA* pCtrls[]);
void fillVectorOfControllerWithLSPBTraj(PIDCtrlPVA* pCtrls[]);
```

The first function fills the vector with controllers using a quintic trajectory, while the second function fills the vector with controllers using the LSPB.

The functions take an array of PIDCtrlPVA pointers, and fill each element with a new instance of a PIDCtrlPVA object that is constructed using a new instance of a TrajPVA object. That is, this is a vector of pointers that points to the PID controller which in the same way, uses pointers to for the trajectory generation.

```
void fillVectorOfControllerWithQuinticTraj(PIDCtrlPVA* pCtrls[])
{
  for (int i = 0; i < 6; i++)
  {
    TrajPVA* pt = new TrajQuinticPVA(posI[i], posF[i], TrajTime, deltaT);
    pCtrls[i] = new PIDCtrlPVA(pt, Kp[i], Ki[i], Kv[i]);

  }
}

void fillVectorOfControllerWithLSPBTraj(PIDCtrlPVA* pCtrls[])
{
  for (int i = 0; i < 6; i++)
  {
    TrajPVA* pt = new TrajLSPBPVA(posI[i], posF[i], velConsts[i], TrajTime, deltaT);
    pCtrls[i] = new PIDCtrlPVA(pt, Kp[i], Ki[i], Kv[i]);
  }
}
```

The main function begins by declaring a TX90ProxyByTorque object, which represents the robotic arm that the program will control. It then creates an array of PIDCtrlPVA pointers to represent the six joints of the robotic arm. It also creates an object of DynamicCompensator class and an object of HystoricData class to save the data.

The fillVectorOfControllerWithQuinticTraj function is then called, which fills the array of PIDCtrlPVA pointers with the TrajQuinticPVA trajectories for each joint.

```
TX90ProxyByTorque robot;
PIDCtrlPVA* pCtrls[6];
DynamicCompesator compensator;
HystoricData hystoricData;

// fills the PID pointers with the needed trayectory
fillVectorOfControllerWithQuinticTraj(pCtrls);
```

The control begins calculating the PID output for each joint. For every iteration of the control, the actual position and the velocity of the robot are read and used by the 6 pCtrls objects to calculate the output torque of the PID.

```
// PID
for (int j = 0; j < 6; j++)
{
  robotCommand[j] = pCtrls[j]->getPidCommand(i, robot.getPosition(j), robot.getSpeed(j));
}
```

For the PID, previously 6pCtrls objects were created from the Class PIDCtrlPVA. This class uses the gains previously initialized *Kp, Ki and Kv*, the previous position error *ErrorAnt*, the integral error *ErrorInt* and a pointer that points to the chosen trayectory, in this case the obtained from the TrajQuinticPVA class.

The the dynamic compensation is calculated. For that, an object "compensator" from class DynamicCompensator is created which takes the actual position and velocity and computes every term of the compensation: $C(q,\dot{q})$, g(q) and fric($\dot{q}$). The firs line refreshes the values of the position and velocity and the second line computes the actual compensation.

```
// position and speed are used by the Dynamic compesator class to calculate the compensation terms C(q,q) and g(q)
compensator.setPosAndVel(robot.getPositions(), robot.getSpeeds());

// compute compensation
compensatedTorque = (compensator.getMassMatrix() * robotCommand) + compensator.getDynamicComp();// + compensator.getFric() * i * 0.04;
```

The hystoricData object from the HystoricData class is used to store the trajectory data, including the joint positions, velocities, and torques.

```
//save data to export
hystoricData.addQr(robot.getPositions());
hystoricData.addDqr(robot.getSpeeds());
hystoricData.addTorque(compensatedTorque);
```

After the loop has completed, the trajectory data is written to a file, and the PIDCtrlPVA pointers are deleted to free up memory.